MOBILE DEVICE HAVING EXTENSIBLE SOFTWARE FOR PRESENTING SERVER-SIDE APPLICATIONS, SOFTWARE AND METHODS

COPYRIGHT NOTICE

FIELD OF THE INVENTION

[0002]   The present invention relates to software, devices and methods allowing varied mobile devices to interact with server side software applications.

BACKGROUND OF THE INVENTION

[0003]   Wireless connectivity is a feature of the modern telecommunications environment. An increasing range of people are using a wide variety of wireless data networks to access corporate data applications.

[0004]   However, there are numerous competing mobile devices that can be used to achieve this. Each device has its own operating system and its own display characteristics. Operating systems are not mutually compatible, nor are the display characteristics--some are color, some are black and white, some are text-only, some are pictorial.

[0005]   At the same time, an increasing number of mobile device users are people without a technical background or high level of educational achievement. Such people are often intimidated by the need to run complex installation

1

programs. Furthermore, at present, such installation programs generally depend on cable connections to a personal computer by the means of a 'cradle' or other such device.

[0006]   U.S. Patent Publication No. US 2003/0060896 discloses a mechanism allowing server-side applications to be presented at multiple wireless devices with minimal modification of the application at the server. As disclosed, how an application is presented at a mobile device is defined by a text based application definition file. The definition file describes how an application is to be presented at the mobile device; the format of transactions over the wireless network; and a format of data related to the application to be stored at the mobile device. A virtual machine software component at the mobile device interprets the definition file and presents an interface to the application in accordance with the definition file. Conveniently, the application definition file may be independent of the particular type of mobile device, while virtual machine software components specific to the mobile device may be created.

[0007]   This approach, while flexible in many ways, is somewhat limited. For example, how an application may be presented at the mobile device, and what resources of the mobile device may be used is limited by the nature of the virtual machine software at the device. The virtual machine software component is typically written with a specific mobile device with specific hardware in mind. As the mobile device is expanded to, for example, include new hardware or local software applications the server side application can typically not take advantages of the new hardware and software. Of course, the virtual machine software component could be rewritten (or recompiled). This, however, is cumbersome and would require many versions of virtual machine software specific to many different hardware configurations.

[0008]   Accordingly, there is a need for virtual machine software that is extensible.

2

SUMMARY OF THE INVENTION

[0009] In accordance with the present invention, data from an application executing at a computing device is presented at a remote wireless device, by providing the device an application definition file, containing a definition of a user interface for the application at the mobile device. Based on the definition file, the wireless device may receive data from the application and present an interface for the application. Virtual machine software at the mobile device interprets the application definition file. This virtual machine software is extensible to takes advantage of other software and/or hardware at the device.

[0010] Preferably, the application definition file is an XML file. Similarly, application specific network messages provided to the device are also formed using XML. Specific XML tags are used to execute software at the mobile device, but external to the virtual machine software. In particular, object classes external to the virtual machine may be instantiated and their methods performed.

[0011] In accordance with an aspect of the present invention, a method of presenting data from an application executing at a computing device at a wireless mobile device remote from the computing device, includes receiving at the mobile device, a representation of a text file defining a user interface and actions to be taken in response to user interaction with the user interface or received data from the application; receiving data from the application; executing virtual machine software at the mobile device to present the user interface and the received data, in accordance with the text file; wherein at least one of the actions in the text file specifies execution of a software component separate from the virtual machine software, identified in the text file and loaded at the device; and executing the software component at the device in response to the at least one of the actions.

[0012] In accordance with another aspect of the present invention wireless mobile device comprising a processor; computer readable memory in communication with the processor, storing virtual machine software controlling operation of the wireless mobile device, the virtual machine software including, a parser for receiving a text file; a screen generation engine, for presenting at least one screen at the wireless mobile device in accordance with the text file; an event

3

handler for processing events arising in response to interaction with the at least one screen in accordance with the text file, the event handler operable to execute a software component separate from the virtual machine software, identified in the text file and loaded at the device.

[0013]   Other aspects and features of the present invention will become apparent to those of ordinary skill in the art upon review of the following description of specific embodiments of the invention in conjunction with the accompanying figures.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014]   In the figures which illustrate by way of example only, embodiments of the present invention,

[0015]   FIG. 1 schematically illustrates a mobile device, exemplary of an embodiment of the present invention, including virtual machine software, further exemplary of an embodiment of the present invention;

[0016]   FIG. 2 further illustrates the organization of exemplary virtual machine software at the mobile device of FIG. 1;

[0017]   FIG. 3 illustrates an operating environment for the device of FIG. 1;

[0018]   FIG. 4 illustrates the structure of example application definitions stored at a server of FIG. 2 used by the device of FIG. 1;

[0019]   FIG. 5 schematically illustrates the formation of application definition files at a middleware server of FIG. 2;

[0020]   FIG. 6 schematically illustrates the middleware server of FIG. 2, exemplary of an embodiment of the present invention, including an application definitions database, further exemplary of an embodiment of the present invention;

[0021]   FIG. 7 is a flow diagram illustrating the exchange of sample messages passed between a mobile device, middleware server and application server of FIG. 2;

[0022]   FIGS. 8-11 illustrate steps performed at a mobile device under control of virtual machine software of FIG. 2;

[0023]   FIG. 12 illustrates the format of messages exchanged in the message flow of FIG. 7;

[0024]   FIGS. 13A and 13B illustrates a presentation of a user interface for a sample application at a mobile device;

[0025]   FIG. 14, 15 and 16A-16B illustrate a sample portion of an application definition file defining a user interface illustrated in FIG. 13; and

[0026]   FIGS. 17A-17PPP contain Appendix "A" detailing example XML entities understood by the virtual machine software of the mobile device of FIG. 1.

DETAILED DESCRIPTION

[0027]   FIG. 1 illustrates a mobile device 10, exemplary of an embodiment of the present invention. Mobile device 10 may be any conventional mobile device, modified to function in manners exemplary of the present invention. As such, mobile device 10 includes a processor 12, in communication with a network interface 14, storage memory 16, and a user interface 18 typically including a keypad and/or touch-screen. Network interface 14 enables device 10 to transmit and receive data over a wireless network 22. Mobile device 10 may be, for example, be a Research in Motion (RIM) two-way paging device, a WinCE based device, a PalmOS device, a WAP enabled mobile telephone, or the like. Memory 16 of device 10 stores a mobile operating system such as the PalmOS, or WinCE operating system software 20. Operating system software 20 typically includes graphical user interface and network interface software having suitable application programmer interfaces ("API"s) for use by other applications executing at device 10.

5

[0028]   Memory at device **10** further stores virtual machine software **24**, exemplary of an embodiment of the present invention. Virtual machine software **24**, when executed by mobile device **10** enables device **10** to present an interface for server side applications provided by a middleware server, described below. Specifically, virtual machine software **24** interprets a text application definition file defining a user interface **18** controlling application functionality, and the display format (including display flow) at device **10** for a particular server-side application; the format of data to be exchanged over the wireless network for the application; and the format of data to be stored locally at device **10** for the application. Virtual machine software **24** uses operating system software **20** and associated APIs to interact with device **10**, in accordance with the received application definition file. In this way, device **10** may present interfaces for a variety of applications, stored at a server. From the perspective of operating system software **20**, virtual machine software **24** is viewed as another application resident at device **10**. Moreover, multiple wireless devices each having a similar virtual machine software **24** may use a common server side application in combination with an application definition, to present a user interface and program flow specifically adapted for the device.

[0029]   As such, and as will become apparent, the exemplary virtual machine software **24** is specifically adapted to work with the particular mobile device **10**. Thus if device **10** is a RIM Blackberry device, virtual machine software **24** is a RIM virtual machine. Similarly, if device **10** is a PalmOS or WinCE device, virtual machine software **24** would be a PalmOS or a WinCE virtual machine. As further illustrated in FIG. **1**, virtual machine software **24** is capable of accessing local storage **26** at device **10**.

[0030]   Other applications, libraries, and software may also be present within memory **16** or local storage **26**, and are not specifically illustrated. For example, device 10 may store and execute personal information management (PIM) software, including calendar and contact management applications. Similarly, device **10** could store and execute software allowing device **10** to perform a number of functions. Software could, for example, interact with the hardware at device **10** to allow device **10** to act as a multimedia player; allowing device **10** to print; allowing device **10** to interact with other incorporated hardware not specifically

6

illustrated, including but not limited to a Bluetooth interface; a Global Positioning Satellite (GPS) Receiver; and the like. In the depicted embodiment, however, memory **16** stores software components in the form of object classes **29** that may be used to extend the functionality of virtual machine software **24**. As will become apparent these external software components in the form of object classes **29** allow virtual machine software **24** to become extensible. Object classes **29** may, for example, allow virtual machine software to access additional hardware or software local to device **10**.

[0031]   As detailed below, an exemplary application definition file may be formed using a mark-up language, like XML. In accordance with an embodiment of the present invention, defined XML entities are understood by the virtual machine software **24**. Defined XML entities are detailed in Appendix "A", hereto and Appendix "A" of U.S. Patent Publication **2003/0060896**. The defined XML entities are interpreted by the virtual machine software **24**, and may be used as building blocks to present server-side applications at mobile device **10**, as detailed herein.

[0032]   Specifically, as illustrated in **FIG. 2**, virtual machine software **24** includes a conventional XML parser **61**; an event handler **65**; a screen generation engine **67**; and object classes **69** corresponding to XML entities supported by the virtual machine software **24**, and possibly contained within an application definition **28**. Supported XML entities are detailed in Appendix "A" hereto. A person of ordinary skill will readily appreciate that those XML entities identified in Appendix "A" are exemplary only, and may be extended, or shortened as desired.

[0033]   XML parser **61** may be formed in accordance with the Document Object Model, or DOM, available at http://www.w3.org/DOM/, the contents of which are hereby incorporated by reference. Parser **61** enables virtual machine software **24** to read an application definition file. Using the parser, the virtual machine software **24** may form a binary representation of the application definition file for storage at the mobile device, thereby eliminating the need to parse text each time an application is used. Parser **61** may convert each XML tag contained in the application definition file, and its associated data to tokens, for later processing. As will become apparent, this may avoid the need to repeatedly parse the text of an application definition file.

7

[0034] Screen generation engine **67** displays initial and subsequent screens at the mobile device, in accordance with an application definition **28**, as detailed below.

[0035] Event handler **65** of virtual machine software **24** allows device **10** under control of virtual machine software **24** to react to certain external events. Example events include user interaction with presented screens or display elements, incoming messages received from a wireless network, or the like.

[0036] Object classes **69** also form part of virtual machine software **24** and define objects that allow device **10** to process each of the supported XML entities at the mobile device. Each of object classes **69** includes attributes used to store parameters defined by the XML file, and functions allowing the XML entity to be processed at the mobile device, as detailed in Appendix "A", for each supported XML entity. So, as should be apparent, supported XML entities are extensible. Virtual machine software **24** may be expanded to support XML entities not detailed in Appendix "A". Corresponding object classes could be added to virtual machine software **24**.

[0037] As detailed below, upon invocation of a particular application at mobile device **10**, the virtual machine software **24** presents an initial screen based on the contents of the application definition **28** for the application. Screen elements are created by screen generation engine **67** by creating instances of corresponding object classes for defined elements, as contained within object classes **69**. The object instances are created using attributes contained in the application definition **28**. Thereafter the event handler **65** of the virtual machine software **24** reacts to events for the application. Again, the event handler consults the contents of the application definition file for the application in order to properly react to events. Events may be reacted to by creating instances of associated "action" objects, from object classes **69** of virtual machine software **24**.

[0038] Similarly, object classes **69** of virtual machine software **24** further include object classes corresponding to data tables and network transactions defined in the Table Definition and Package Definition sections of Appendix "A". At run time,

instances of object classes corresponding to these classes are created and populated with parameters contained within application definition file, as required.

[0039] Using this general description, persons of ordinary skill in the art will be able to form virtual machine software **24** for any particular device. Typically, virtual machine software **24** may be formed using conventional object oriented programming techniques, and existing device libraries and APIs, as to function as detailed herein. As will be appreciated, the particular format of screen generation engine **67**, object classes **69** will vary depending on the type of virtual machine software, its operating system and API available at the device. Once formed, a machine executable version of virtual machine software **24** may be loaded and stored at a mobile device, using conventional techniques. It can be embedded in ROM, loaded into RAM over a network, or from a computer readable medium.

[0040] As so far described, however, operation of virtual machine software **24** is limited by those object classes **69** forming part of virtual machine software **24**. However, exemplary of embodiments of the present invention object classes **29** not forming part of virtual machine software **24**, are further loaded within memory 16 of device **10**. Conveniently, object classes **29** may be created by a user (or administrator) of device **10** do not rely on access to the source code for virtual machine software **24**.

[0041] Instead, as will become apparent, virtual machine software **24** includes a software code portion that instantiates identified ones of object classes **29**, external to virtual machine software **24** and executes methods contained in the object classes **29**. As such, virtual machine software **24** may be extended through the addition of additional object classes.

[0042] Although, in the preferred embodiment the virtual machine software **24** and software forming object classes **29** are formed using object oriented structures, persons of ordinary skill will readily appreciate that other approaches could be used to form suitable virtual machine software. For example, object classes **69** forming part of the virtual machine could be replaced by equivalent functions, data structures or subroutines formed using a conventional (i.e. non-object oriented) programming environment. Object classes **29** could be similarly replaced with

9

other software components in the form of libraries, sub-routines, programs, combinations thereof, or the like.

[0043]   Operation of virtual machine software **24** under control of an application definition containing various XML definitions exemplified in Appendix "A" is further detailed below.

[0044]   **FIG. 3** illustrates the operating environment for a mobile device **10**. Further example mobile devices **30**, **32** and **34** are also illustrated in **FIG. 3**. These mobile devices **30**, **32** and **34** are similar to device **10** and also store and execute virtual machine software exemplary of an embodiment of the present invention.

[0045]   Virtual machine software like that stored at device **10**, executes on each mobile device **10**, **30**, **32**, **34**, and communicates with a middleware server **44** by way of example wireless networks **36** and **38** and network gateways **40** and **42**. Example gateways **40** and **42** are generally available as a service for those people wishing to have data access to wireless networks. An example network gateway is available from Broadbeam Corporation in association with the trademark SystemsGo!. Wireless networks **36** and **38** are further connected to one or more computer data networks, such as the Internet and/or private data networks by way of gateway **40** or **42**. As will be appreciated, the invention may work with many types of wireless networks. Middleware server **44** is in turn in communication with a data network, that is in communication with wireless network **36** and **38**. The communication used for such communication is via TCP/IP over an HTTP, COM or .NET transport. As could be appreciated, other network protocols such as X.25 or SNA could equally be used for this purpose.

[0046]   Devices **10**, **30**, **32**, and **34** communicate with middleware server **44** in two ways. First, virtual machine software **24** at each device may query middleware server **44** for a list of applications that a user of an associated mobile device **10**, **30**, **32** or **34** can make use of. If a user decides to use a particular application, device **10**, **30**, **32** or **34** can download a text description, in the form of an application definition file, for the application from the middleware server **44** over its wireless interface. As noted, the text description is preferably formatted using XML. Second, virtual machine software **24** may send, receive, present, and locally store data

10

related to the execution of applications, or its own internal operations. The format of exchanged data for each application is defined by an associated application definition file. Again, the exchanged data is formatted using XML, in accordance with the application definition file.

[0047] Middleware server **44**, in turn, stores text application definition files for those applications that have been enabled to work with the various devices **10, 30, 32**, and **34** using virtual machine software **24** in a pre-defined format understood by virtual machine software **24**. Software providing the functions of the middleware server **44**, in the exemplary embodiment is written in C#, using an SQL Server or MySQL database.

[0048] As noted, text files defining application definitions and data may be formatted in XML. For example XML version **1.0**, detailed in the XML version **1.0** specification third edition, dated Feb. **4, 2004** and available at the internet address " http://www.w3.org/TR/2004/REC-xml-20040204/ ", the contents of which are hereby incorporated herein by reference, may be used. However, as will be appreciated by those of ordinary skill in the art, the functionality of storing XML description files is not dependent on the use of any given programming language or database system.

[0049] Each file containing an application definition **28** is formatted according to defined rules and uses pre-determined XML mark-up tags known by both virtual machine software **24**, and complementary middleware server software **68**. Tags define XML entities used as building blocks to present an application at a mobile device. Knowledge of these rules, and an understanding of how each tag and section of text should be interpreted, allows virtual machine software **24** to process an XML application definition and thereafter execute an application, as described below. Virtual machine software **24** effectively acts as an interpreter for a given application definition file.

[0050] FIG. **4** illustrates an example format for an XML application definition **28**. As illustrated, the example application definition **28** for a given device and application includes three components: a user interface definition section **48**, specific to the user interface for the device **10**, and defining the format of screen or

11

screens for the application and how the user interacts with them; a network transactions definition section **50** defining the format of data to be exchanged with the application; and a local data definition section **52** defining the format of data to be stored locally on the mobile device by the application.

[0051] Defined XML mark-up tags correspond to XML entities supported at a device, and are used to create an application definition **28**. The defined tags may broadly be classified into three categories, corresponding to the three sections **48**, **50** and **52** of an application definition **28**.

[0052] Example XML tags and their corresponding significance are detailed in Appendix "A". As noted above, virtual machine software **24** at a mobile device includes object classes corresponding to each of the XML tags. At run time, instances of the objects are created as required.

[0053] Broadly, the following example XML tags may be used to define the user interface definition:

SCREEN -- this defines a screen. A SCREEN tag pair contains the definitions of the user interface elements (buftons, radio buttons, and the like) and the events associated with the screen and its elements

BUTTON -- this tag defines a button and its associated attributes

LIST -- this tag defines a list box

CHOICEBOX -- this tag defines a choice item, that allows selection of a value from predefined list

MENU -- the application developer will use this tag to define a menu for a given screen

EDITBOX -- this tag defines an edit box

TEXT ITEM -- this tag describes a text label that is displayed

12

CHECKBOX -- this tag describes a checkbox

HELP -- this tag can define a help topic that is used by another element on the screen

IMAGE -- this tag describes an image that appears on those displays that support images

ICON -- this tag describes an icon

EVENT -- this defines an event to be processed by the virtual machine software. Events can be defined against the application as a whole, individual screens or individual items on a given screen. Sample events would be receipt of data over the wireless interface, or a edit of text in an edit box

ACTION -- this describes a particular action that might be associated with an event handler. Sample actions would be navigating to a new window or displaying a message box.

[0054]   The second category of example XML tags describes the network transaction section **50** of application definition **28**. These may include the following example XML tags;

TABLEUPDATE--using this tag, the application developer can define an update that is performed to a table in the device's local storage. Attributes allow the update to be performed against multiple rows in a given table at once;

PACKAGEFIELD--this tag is used to define a field in a data package that passes over the wireless interface

[0055] The third category of XML tags used to describe an application are those used to define a logical database that may be stored at the mobile device. The tags available that may be used in this section are:

TABLE--this tag, and its attributes, define a table. Contained within a pair of TABLE tags are definitions of the fields contained in that table. The attributes of a table control such standard relational database functions as the primary key for the table.

FIELD--this tag describes a field and its attributes. Attributes of a field are those found in a standard relational database system, such as the data type, whether the field relates to one in a different table, the need to index the field, and so on.

[0056] As well as these XML tags, virtual machine software **24** may, from time to time, need to perform certain administrative functions on behalf of a user. In order to do this, one of object classes **69** has its own repertoire of tags to communicate its needs to the middleware server **44**. Such tags differ from the previous three groupings in that they do not form part of an application definition file, but are solely used for administrative communications between the virtual machine software **24** and the middleware server **44**. These tags are again detailed in U.S. Patent Publication US 2003/0060896.

[0057] One specific type of ACTION understood by virtual machine software **24** is referred to as a "INTEGRATION" action. This action is identified as an ACTION having an ACTION TYPE="INTEGRATION" tag, as more particularly exemplified in FIG. xx.

[0058] Specifically, the format of the TYPE tag identifying the INTEGRATION action takes the form

<ACTION TYPE ="INTEGRATION" CLSID = "*class_name*"
SAVENAME="*returnvar*" SAVE="*true/false*"> *my input text*</ACTION>

[0059] That is, the INTEGRATION tag takes as arguments, the name of an external one of classes **29** to be instantiated assigned to CLSID=*class_name*, and the name of a local variable, *returnvar*, used by virtual machine software **24** in

14

which results passed by the execution should be stored. As well, the value assigned to SAVE is boolean and specifies whether or not the data returned by the instantiated class should be saved.

[0060]   Notably, *class_name* identifies one of classes **29** by name. The name of the class is assigned as described below. The name of the local variable corresponds to the name of a variable defined in section **52** of the application definition **28**. Finally, the contents of the ACTION element (i.e. *my input text*) is passed to the instantiated one of classes **29**, as detailed below.

[0061]   The exact way in which external accessible objects are formed and may be accessed by virtual machine software **24** will typically depend on the operating system software **20** of device **10** in association with which virtual machine software **24** is executing. Virtual machine software **24** should, however, be able to identify the external object and instantiate it. Optionally, virtual machine software **24** should be able to verify that the external object class has an interface that conforms to virtual machine software **24**.

[0062]   For example, virtual machine software **24** written and executing in a WindowsCE environment, object classes developed using the component object model (COM) may be accessible. The component object model (COM) is described at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/componentobjectmodelanchor.asp and D. Box, Essential COM, (**1997**: Addison-Wesley Professional) ISBN: **0201634465**, the contents of which are hereby incorporated by reference.

[0063]   Briefly, object classes developed using COM are registered with the WindowsCE operating system. The operating system maintains a list of the COM objects that have been created. Additionally, object classes developed in accordance with the COM include one or more defined interface.

[0064]   Other operating systems executing on mobile devices expose classes that may be accessible by virtual machine software **24** in different ways. For example, RIM and PalmOS operating systems expose various PIM object stores as Java Classes or C++ classes. A person of ordinary skill will readily appreciate how

such classes may be used by virtual machine software **24** created for such an operating system.

[0065] Object classes **29** written in accordance with the COM, exemplary of embodiments of the present invention may register their name with the underlying operating system, and further include an interface. In the preferred embodiment, the interface takes has a name known by virtual machine software **24**. For example, the interface may take the name IAIRIXIntegrationPoint. In the preferred embodiment the interface defines a function with name HRESULT that takes parameter hWndParent, InputString, and OutputString. Variables InputString, and OutputString are populated by values passed virtual machine software **24** identifying attributes of the *string* SAVE_NAME.

[0066] The value of hWndParent identifies the main window generated by virtual machine software **24** as a result of the application definition file instantiating the object class. The value may be used by the method of the instantiated class **29** to embed controls on or as a parent window to sub windows that the method creates.

[0067] To summarize, the interface takes the form

```
interface IAIRIXIntegrationPoint : IDispatch
{
    [id(1), helpstring("method Process")] HRESULT Process(VARIANT
    hWndParent, BSTR InputString, BSTR* OutputString);
};
```

As will be appreciated, IDispatch signifies a standard COM interface; id(1) signifies that the method Process is listed as the first method exposed on the interface; and helpstring may be used by a debugging tool.

[0068] The method Process, in turn, performs the function to be implementd by the external object class to perform the desired functionality, extending the functions performed by virtual machine software **24**. For example, method "Process" could provide an interface to other object classes, or hardware at device **10**. The method "Process" could for example gather a signature, a fingerprint, GPS co-ordinates, or virtually any other function that can be performed by device **10**. Conveniently, the method "Process" may make use of the string data contained as

16

my input text and forming part of the XML element giving rise to the instantiating of the class.

[0069]   Upon completion of the method, results should be formatted by the method and placed in the variable OutputString, so the results may be passed back to virtual machine software **24** for further processing. In the exemplary embodiment, the content of OutputString is XML formatted, so that it may be easily further processed by machine software **24** (or alternatively middleware server **44**).

[0070]   The value of Process returned by the method call may identify successful execution of the method. In response to an error code, virtual machine software **24** may log an error and report that error to the user of device **10** through a standard error message dialog.

[0071]   As will be apparent, the name of each class **29** is identified in the PROGID variable used as each class is created and will be registered in accordance with COM.

[0072]   **FIG. 5** illustrates the organization of application definitions at middleware server **44** and how middleware server **44** may form an application definition **28** (FIG. 4) for a given device **10, 30, 32** or **34**. In the illustration of **FIG. 5**, only two mobile devices **10** and **30** are considered. Typically, since network transactions and local data are the same across devices, the only piece of the application definition that varies for different devices is the user interface definition.

[0073]   So, middleware server **44** stores a master definition **58** for a given server side application. This master definition **58** contains example user interface descriptions **48, 54, 56** for each possible mobile device **10, 30, 32**; descriptions of the network transactions **50** that are possible and data descriptions **52** of the data to be stored locally on the mobile device. Preferably, the network transactions **50** and data descriptions **52** will be the same for all mobile devices **10, 30** and **32**.

[0074]   For device **10**, middleware server **44** composes an application definition **28** by querying the device type and adding an appropriate user interface description **48** for device **10** to the definitions for the network transactions **50** and the data **52**. For device **30**, middleware server **44** composes the application definition by adding

the user interface description **54** for device **10** to the definitions for the network transactions **50** and data **52**.

[0075]   The master definition **58** for a given application is created away from the middleware server and loaded onto the middleware server by administrative staff charged with its operation. Master definition files could be created either by use of a simple text editor, or by a graphical file generation tool. Such a tool might generate part or all of the file, using knowledge of the XML formatting rules, based on the user's interaction with screen painters, graphical data definition tools and the like.

[0076]   FIG. 6 illustrates the organization of middleware server **44** and associated master definitions. Middleware server **44** may be any conventional application server, modified to function in manners exemplary of the present invention. As such, middleware server **44** includes a processor **60**, in communication with a network interface **66** and storage memory **64**. Middleware server **44** may be, for example, be a Windows NT server, a Sun Solaris server, or the like. Memory of middleware server **44** stores an operating system such as Windows NT, or Solaris operating system software **62**.

[0077]   Network interface **66** enables middleware server **44** to transmit and receive data over a data network **63**. Transmissions are used to communicate with both the virtual machine software **24** (via the wireless networks **36, 38** and wireless gateways **40,42**) and one or more application servers, such as application server **70**, that are the end recipients of data sent from the mobile client applications and the generators of data that is sent to the mobile client applications.

[0078]   Memory at middleware server **44** further stores software **68**, exemplary of an embodiment of the present invention. Middleware server software **68**, when executed by middleware server **44** enables the middleware server to understand and compose XML data packages that are sent and received by the middleware server. These packages may be exchanged between middleware server **44** and the virtual machine software **24**, or between the middleware server **44** and the application server **70**.

[0079]   As described above, communication between the application server **70** and the middleware server **44** uses HTTP running on top of a standard TCP/IP

stack. An HTTP connection between a running application at the application server 70 and the middleware server 44 is established in response to the application at a mobile device presenting the application. The server side application provides output to middleware server 44 over this connection. The server side application data is formatted into appropriate XML data packages understood by the virtual machine software 24 at a mobile device by the server side application.

[0080]   That is, a server side application (or an interface portion of the application) formats application output into XML in a manner consistent with the format defined by the application definition file for the application. Alternatively, an interface component separate from the application could easily be formed with an understanding of the format and output for a particular application. That is, with a knowledge of the format of data provided and expected by an application at application server 70, an interface component could be a produced using techniques readily understood by those of ordinary skill. The interface portion could translate application output to XML, as expected by middleware server 44. Similarly, the interface portion may translate XML input from a mobile device into a format understood by the server side application.

[0081]   The particular identity of the mobile device on which the application is to be presented may be identified by a suitable identifier, in the form of a header contained in the server side application output. This header may be used by middleware server 44 to forward the data to the appropriate mobile device. Alternatively, the identity of the connection could be used to forward the data to the appropriate mobile device.

[0082]   FIG. 7 illustrates a flow diagram detailing data (application data or application definition files 28) flow between mobile device 10 and middleware server 44, in manners exemplary of an embodiment of the present invention.

[0083]   For data requested from middleware server 44, device 10, under software control by virtual machine software 24 makes requests to middleware server 44 (also illustrated in FIG. 2), which passes over the wireless network 36 through network gateway 40. Network gateway 40 passes the request to the middleware server 44. Middleware server 44 responds by executing a database

query on its database **46** that finds which applications are available to the user and the user's mobile device. For data passed from middleware server **44** to device **10**, data is routed through network gateway **40**. Network gateway **40** forwards the information to the user's mobile device over the wireless network **36**.

[0084]   **FIG. 7** when considered with **FIG. 3** illustrates a sequence of communications between device **10**, and middleware server **44** that may occur when the user of a mobile device wishes to download an application definition **28** for a server side application.

[0085]   So, initially, device **10** interrogates server **44** to determine which applications are available for the particular mobile device being used. In response, a user at device **10** may choose to register for an available server side application Thereafter the middleware server creates the application definition file and sends to the mobile device (data flow **78**) the created application definition **28**. This all further detailed in US Patent Publication No. US 2003/0060896.

[0086]   The user is then able to use the functionality defined by the interface description to send and receive data.

[0087]   At this time, parser **61** of virtual machine software **24** may parse the XML text of the application definition file to form a tokenized version of the file. That is, each XML tag may be converted a defined token for compact storage, and to minimize repeated parsing of the XML text file. The tokenized version of the application definition file may be stored for immediate or later use by device **10**.

[0088]   Thereafter, upon invocation of a particular application for which the device **10** has registered, the screen generation engine **67** of the virtual machine software **24** at the device causes the virtual device to locate the definition of an initial screen for that application. The initial screen is identified within the application definition **28** for that application using a <SCREEN> tag, and an associated attribute of <First screen="yes">.

[0089]   Steps performed by virtual machine software **24** in processing this screen (and any screen) are illustrated in **FIG. 8**. As illustrated, screen generation engine **67**, generates an instance of an object class, defining a screen by parsing the

section of the XML application definition file corresponding to the <SCREEN> tag in step **S802**. Supported screen elements may be buttons, edit boxes, menus, list boxes, and choice items, as identified in sections **5.3**, **5.4**, and **5.5** of Appendix "A". Other screen elements, such as images and checkboxes, as detailed in Appendix "A" may also be supported. For clarity of illustration, their processing by screen generation engine **67** however, is not detailed. Each supported tag under the SCREEN definition section, in turn causes creation of instances of object classes within the virtual machine software **24**. Typically, instances of objects corresponding to the tags, used for creation of a screen, result in presentation of data at mobile device **10**. As well the creation of such objects may give rise to events (e.g. user interaction) and actions to be processed at device **10**.

[0090] Each element definition causes virtual machine software **24** to use the operating system of the mobile device to create corresponding display element of a graphical user interface as more particularly illustrated in **FIG. 9**. Specifically, for each element, the associated XML definition is read in step **S806**, **S816**, **S826**, **S836**, and **S846**, and a corresponding instance of a screen object defined as part of the virtual machine software **24** is created by the virtual machine software **24** in steps **S808**, **S818**, **S828**, **S838** and **S848**, in accordance with steps **S902** and onward illustrated in FIG. **9**. Each interface object instance is created in step **S902**. Each instance takes as attribute values defined by the XML text associated with the element. A method of the virtual object is further called in step **S904**, and causes a corresponding device operating system object to be created. Those attributes defined in the XML text file, stored within the virtual machine object instance are applied to the corresponding instance of a display object created using the device operating system in steps **S908S-S914**. These steps are repeated for all attributes of the virtual machine object instance. For any element allowing user interaction, giving rise to an operating system event, the event handler **65** of virtual machine software **24** is registered to process operating system events, as detailed below.

[0091] Additionally, for each event (as identified by an <EVENT> tag) and action (as identified by an <ACTION> tag) associated with each XML element, virtual machine software **24** creates an instance of a corresponding event and action object forming part of virtual machine software **24**. Virtual machine software **24**

further maintains a list identifying each instance of each event and action object, and an associated identifier of an event in steps **S916** to **S928**.

[0092]   Steps **S902-S930** are repeated for each element of the screen in steps **S808, S818, S828, S838** and **S848** as illustrated in FIG. **8**. All elements between the <SCREEN> definition tags are so processed. After the entire screen has been so created in memory, it is displayed in step **S854**, using conventional techniques.

[0093]   As will be appreciated, objects specific to the type of device executing the virtual machine software **24**. Functions initiated as a result of the XML description may require event handling. This event handling is processed by event handler **65** of virtual machine software **24** in accordance with the application definition **28**. Similarly, receipt of data from a mobile network will give rise to events. Event handler **65**, associated with a particular application presented at the device similarly processes incoming messages for that particular application. In response to the events, virtual machine software **24** creates instance of software objects, and calls functions of those object instances, as required by the definitions contained within the XML definitions contained within the application definition **28**, giving rise to the event.

[0094]   As noted, the virtual machine software **24** includes object classes, allowing the virtual machine to create object instances corresponding to an <EVENT> tag. The event object classes includes methods specific to the mobile device that allow the device to process each of the defined XML descriptions contained within the application definition file, and also to process program/event flow resulting from the processing of each XML description.

[0095]   Events may be handled by virtual machine software **24** as illustrated in FIG. **10**. Specifically, as device handler **65** has been registered with the operating system for created objects, upon occurrence of an event, steps **S1002** and onward are performed in response to the operating system detecting an event.

[0096]   An identifier of the event is passed to event handler **65** in step S1002. In steps **S1004-S1008**, this identifier is compared to the known list of events, created as a result of steps **S916-S930**. For an identified event, actions associated with that event are processed in step **S1008-S1014**.

22

[0097]   That is, virtual machine software **24** performs the action defined as a result of the <ACTION> tag associated with the <EVENT> tag corresponding to the event giving rise to processing by the event handler **65**. The <ACTION> may cause creation of a new screen, as defined by a screen tag, a network transmission, a local storage, or the like.

[0098]   In particular, if the NAME tag associated with the action identifies INTEGRATION tag, virtual machine software **24** performs steps **S1100** depicted in **FIG. 11**, as a result of executing the next action in step **S1012**. As illustrated, virtual machine software **24** compares the value provided to the CLSID variable to the names of accessible classes not forming part of virtual machine software **24**.

[0099]   In step **S1102**, virtual machine software **24** queries the list of available accessible object classes. If objects classes **29** were created using COM, as detailed above, this is accomplished by querying the WindowsCE registry. If an object class corresponding to the class identified in the CLID (i.e. CLID=class_name) variable is found as determined in steps **S1104**, the class is queried in step **S1106**, to verify that the class indeed extends virtual machine software **24**. Querying may be accomplished by querying the class to determine if it provides the interface expected by virtual machine software **24**. Specifically, the class may be queried to determine if it has an interface having a chose name or type. The interface may be queried using the COM method QueryInterface(). In the example the object class is queried to locate an interface having the name IAIRIXIntegrationPoint is used. If the class does not have the interface, the INTEGRATION action is terminated by machine software **24** and an error message could optionally be generated.

[00100]   If the class has the interface, the class is instantiated in step **S1112** and a method having a chosen name (i.e. Process) is executed by virtual machine software **24** in step **S1114**. Parameters hWndParent and the input and output strings formed (i.e. *my input text, returnvar* passed to variable SAVENAME) part of the tag and XML element are passed to the method. As noted, the actual function of the method is entirely determined by the author of the class, and not the provider of virtual machine software **24**. Upon completion of the executed method, the results of the method are passed to virtual machine software **24**, by assigning the

result to the variable *OutputString*. If the SAVE variable is set to true, the results returned by the method are stored in the variable identified assigned to the SAVENAME variable in step **S1116**. If the identified class does not include the expected interface as determined in step **S1110**, the INTEGRATION action is terminated. Again, an error message could be generated.

[00101]   Conveniently, once data returned by the method call is stored locally in a variable defined in application definition **28** and otherwise accessible by virtual machine software **24**.   Of course, the contents of the variable may be acted upon as otherwise dictated by the application definition **28**. Thus, contents of the variable may be presented as part of the user interface, or sent back to middleware server **44**, for example as part of a message defined in portion **50** of the application definition 28 as identified by the <DATA> tag, as detailed in Appendix "A".

[00102]   After execution of the method of the external class **29**, additional screens, may be created by invocation of the screen generation engine **67**, as detailed in **FIGS. 8** and **9**. In this manner the navigation through the screens of the application is accomplished according to the definition embodied in the XML application definition.

[00103]   Similarly, when the user wishes to communicate with the middleware server, or store data locally, event handler **65** creates instances of corresponding object classes within the object classes **69** of virtual machine software **24** and calls their methods to store or transmit the data using the local device operating system. The format of data is defined by the device local definition section **52**; the format of network packages is defined in the network transaction package definition section **50**.

[00104]   For example, data that is to be sent to the wireless network is assembled into the correct XML packages using methods within an XML builder object, formed as a result of creating an instance of a corresponding object class within object classes **69** of virtual machine software **24**. Methods of the XML builder object create a full XML package before passing the completed XML package to another message server object. The message server object uses the device's

network APIs to transmits the assembled data package across the wireless network.

[00105]   Received XML data packages from network 63 (FIG. 2) give rise to events processed by event handler 65. Processing of the receipt of data packages is not specifically illustrated in FIG. 9. However, the receipt of data triggers a "data" event of the mobile device's operating system. This data event is passed to the virtual machine, and event handler 65 inspects the package received. As long as the data received is a valid XML data package as contained within the application definition, the virtual machine inspects the list of recognised XML entities.

[00106]   So, for example, a user could send a login request 80 by interacting with an initial login screen, defined in the application definition file for the application. This would be passed by the middleware server 44 to the backend application server 70. The backend application server according to the logic embedded within its application, would return a response, which the middleware server 44 would pass to the virtual machine software 24. Other applications, running on the same or other application servers might involve different interactions, the nature of such interactions being solely dependent on the functionality and logic embedded within the application server 70, and remaining independent of the middleware server 44.

[00107]   FIG. 12 illustrates sample XML messages passed as the result of message flows illustrated in FIG. 6. For each message, the header portion, between the <HEAD> . . . </HEAD> tags contains a timestamp and the identifier of the sending device.

[00108]   Example message 72 is sent by the mobile device to request the list of applications that the server has available to that user on that device. It specifies the type of device by a text ID contained between the <PLATFORM> . . . </PLATFORM> tags. Example message 74 is sent in response to message 70 by middleware server 44 to the mobile device 10. It contains a set of <APP> . . . </APP> tag pairs, each of which identifying a single application that is available to the user at device 10. Example message 76 is sent from the mobile device 10 to middleware server 44 to register for a single server side application. The tags

specify information about the user. Message **78** is sent by the middleware server **44** to the mobile device in response to a request to register device **10** for an application. The pair of tags <VALUE> . . . </VALUE> gives a code indicating success or failure. In the sample message shown, a success is shown, and is followed by the interface description for the application, contained between the <INTERFACE> . . . </INTERFACE> tags. This interface description may then be stored locally within memory **16** of device **10**.

[00109]   As noted, when a user starts an application that has been downloaded in the manner described above, the virtual machine software **24** reads the interface description that was downloaded for that device **10**, and the virtual machine software **24** identifies the screen that should be displayed on startup, and displays its elements as detailed in relation to **FIGS. 9** and **10**. The user may then use the functionality defined by the user interface definition section **48** of the application definition **28** to send and receive data from a server side application.

[00110]   For the purposes of illustration, **FIGS. 13** illustrates the presentation of a user interface for a sample screen on a Windows CE Portable Digital Assistant, that has invoked an externally generated signature capture dialog, as a result of an externally instantiated object. The signature data is stored in a variable LASTSIG, and sent back to application server **44**.

[00111]   An example application definition file **92** (of the format of application definition **28**) is illustrated in **FIGS. 14, 15** and **16A-16B** defines an application entitled SignatureCapture, including a definition of local data in the form of a table titled LASTSIG (**FIG. 14**), a format of a user interface having a single screen entitled MAIN (**FIG. 16A-16B**) and the format of network transactions (**FIG. 15**), corresponding to portions **52, 50** and **52**, respectively of an application definition **28**.

[00112]   The screen has four single buttons identified by the 'BTN NAME'=" **btnCapture**", BTN NAME'="**btnView**", 'BTN NAME'="**btnSend**", BTN NAME="**btnClose**".

[00113]   Upon invocation of this application at the local device, screen generation engine **67** of virtual machine software **24** at the device process the

screen definition, as detailed with reference to **FIGS. 8** and **9**. That is, for each BTN tag, screen generation engine **67** creates a button object instance, in accordance with steps **S804-S812**. The created buttons will have captions Capture New Signature, View Last Signature, Send to Server, and Close.

[00114]   The resulting screen at the mobile device is illustrated in **FIG. 13A**. Each of the screen items is identified with reference to the XML segment within XML definition file **92** giving rise to the screen element.

[00115]   Call-backs associated with the presented buttons cause graphical user interface application software/operating system software **20** at the mobile device to return control to the event handler **65** of virtual machine software **24** at the device. Thus, as the user interacts with the application, the user may input data within the presented screen using the mobile device API.

[00116]   Notably, if the button btnSend is pressed, a package of type SIG (as defined in **FIG. 15**) is sent back to middleware server **44**. If btnClose is pressed, the application is closed.

[00117]   However, if the buttons btnCapture or btnView are captured steps **S1100** are performed to instantiate an external object class **29** named AirixSignature.AirixSignatureCtrl, with arguments,

SAVENAME="SIGNATURE" SAVE="TRUE" for btnCapture, and

SAVENAME="" SAVE="FALSE">[SP.*.SIGNATURE]</ACTION>, for btnView.

[00118]   An object class **29** named AirixSignature.AirixSignatureCtrl, of course needs to exist, be registered and expose an interface of the forme IAIRIXIntegrationPoint, as detailed above. Its method Process, in turn, causes device **10** to capture a signature or present the signature. These functions may for example be provided using software written for the WindowsCE platform, in a manner appreciated by a person of ordinary skill. In the case of the btnCapture INTEGRATION action, the results of the method return a captured signature, which is stored in variable SIGNATURE by virtual machine software **24**. In the case of the btnView the previously captured value stored in the variable SIGNATURE will be passed to the instance of the object class.

27

[00119]   The screen presented at device **10** in response to performing the Process method of the AirixSignature.AirixSignatureCtrl object class is displayed in **FIG. 13B.**

[00120]   As can be appreciated from the preceding description and example, use of external software components in the form of object classes **29** allows virtual machine software **24** to be expanded, almost arbitrarily. Conveniently, applications may still be defined using an application definition in a manner relatively abstracted from the underlying device. Conveniently, as local software or hardware functions are added to devices, virtual machine software **24** and applications defined in an application definition **28** may take advantage of the new functionality using external object classes **29.**

[00121]   It will be further understood that the invention is not limited to the embodiments described herein which are merely illustrative of a preferred embodiment of carrying out the invention, and which is susceptible to modification of form, arrangement of parts, steps, details and order of operation. The invention, rather, is intended to encompass all such modification within its scope, as defined by the claims.